River Crossing — Wolf, Sheep, and Cabbage

Dr Alun Moon

5th February 2021

1 Requirements

> A farmer is on the bank of a river with a wolf, a sheep and a cabbage. He wants to cross the river, taking with him the wolf, sheep, and cabbage. There is a boat that can fit the farmer plus one of the wolf, the sheep, or the cabbage. However, if the wolf and the sheep are ever left on the same bank without the farmer, the wolf will eat the sheep. If the sheep and the cabbage are ever left on the same bank without the farmer, the sheep will eat the cabbage. How can the farmer safely transport the wolf, the sheep, and the cabbage so that all of them are eventually on the other side of the river together?

2 Discussion

2.1 State

River banks The first step is that the river banks can be modelled as sets, containing the *Wolf*, *Sheep*, and *Cabbage*. We will need a some way of determining which pairs cannot be left together (some state predicate).

Type Invariant The river bank sets will be some subset of the initial collection. If we set a constant from the model containing the wolf, sheep, and cabbage; it makes the problem more general (hence useful). The banks will be some subset of this model set.

Boat and Farmer the boat and farmer can be abstracted out in some way. We are only interested in moving one thing between the banks, the boat and farmer are implicitly moved. We need to keep track of the boat contents and its location. We can use a record for this.

Type invariant The boat can be a record with a field for the passenger (an item from the constant set), and a location which will be the left or right bank.

The Initial conditions are that everything starts at the left bank.

Safety Condition We can't leave the Wolf and Sheep together, or the Sheep and Cabbage together. We need some state predicate to say if a bank is safe to be left with it's occupants. We can either test the state directly, or have an operator that tests if one thing eats another. *I'll go with the operator as this illustrates more of TLA+*

Eats We want an operator that is true, if one thing eats another. The model can supply a set of pairs (tuples) of things that eat another. We can test if the parameters given to the operator form a pair in the model.

Bank is Safe if there is no predator that can eat some prey.

$$Safe(bank) \triangleq \neg\exists\, predator \in bank : \exists\, prey \in bank : Eats(predator, prey)$$

Actions As a first look there are the following actions to consider.

1. moving something from one bank to another
2. moving the boat while empty
3. loading the boat
4. unloading the boat

Loading and Unloading the boat can be assumed to be part of moving one thing across the river.

An action changes the state of the system. Consider the change that occurs,

1. One thing is now not on one bank
2. That same thing is now on the other bank
3. The boat has changed position

We can write an operator that makes these changes.

Left-to-Right and Right-to-Left We can now write some more explicit operators that handle individual moves. We can move something from the left-bank to the right-bank, if the boat is at the left bank, and the left-bank is left in a safe condition. *Visa-versa* for the right-bank.

Empty move Moving the boat when empty, changes the bank the boat is at, while leaving the left and right bank unchanged and in safe conditions..

Next action The next action is a simple disjunction (OR) of moves where the move is:

- an empty move
- something can be moved from left to right banks
- something can be moved from right to left banks

Complete The puzzle is complete when the left-bank is empty and the right-bank is the model set.

2.2 Limitations

3 Specification

MODULE *RiverCrossing*

CONSTANTS *Farmyard*,
Predates

VARIABLES *Leftbank*,
Rightbank,
Boat

$$
\begin{aligned}
&\mathit{TypeInvariant} \triangleq \\
&\quad \land \mathit{Leftbank} \subseteq \mathit{Farmyard} \\
&\quad \land \mathit{Rightbank} \subseteq \mathit{Farmyard} \\
&\quad \land \mathit{Boat} \in \{\text{“left”}, \text{“right”}\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{Init} \triangleq \\
&\quad \land \mathit{Leftbank} = \mathit{Farmyard} \\
&\quad \land \mathit{Rightbank} = \{\} \\
&\quad \land \mathit{Boat} = \text{“left”}
\end{aligned}
$$

$$\mathit{Eats}(\mathit{predator}, \mathit{prey}) \triangleq \langle \mathit{predator}, \mathit{prey} \rangle \in \mathit{Predates}$$

$$\mathit{Opposite}(\mathit{bank}) \triangleq \text{IF } \mathit{bank} = \text{“left”} \text{ THEN “right” ELSE “left”}$$

$$
\begin{aligned}
&\mathit{Safe}(\mathit{bank}) \triangleq \neg\exists\, \mathit{predator} \in \mathit{bank} : \\
&\qquad\qquad \exists\, \mathit{prey} \in \mathit{bank} : \mathit{Eats}(\mathit{predator}, \mathit{prey})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{Move}(\mathit{thing}, \mathit{From}, \mathit{To}) \triangleq \\
&\quad \land \mathit{From}' = \mathit{From} \setminus \{\mathit{thing}\} \\
&\quad \land \mathit{To}' \quad = \mathit{To} \cup \{\mathit{thing}\} \\
&\quad \land \mathit{Boat}' \;= \mathit{Opposite}(\mathit{Boat})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{MoveLeft}(\mathit{thing}) \triangleq \\
&\quad \land \mathit{Boat} = \text{“left”} \\
&\quad \land \mathit{Move}(\mathit{thing}, \mathit{Leftbank}, \mathit{Rightbank})
\end{aligned}
$$

$\land\ Safe(Leftbank')$

$MoveRight(thing) \triangleq$
$\land\ Boat = \text{“right”}$
$\land\ Move(thing,\ Rightbank,\ Leftbank)$
$\land\ Safe(Rightbank')$

$MoveEmpty \triangleq$
$\land\ Boat' = Opposite(Boat)$
$\land$ UNCHANGED $Leftbank \land Safe(Leftbank)$
$\land$ UNCHANGED $Rightbank \land Safe(Rightbank)$

$Next \triangleq$
$\lor\ MoveEmpty$
$\lor \exists\, thing \in Leftbank : MoveLeft(thing)$
$\lor \exists\, thing \in Rightbank : MoveRight(thing)$

$Complete \triangleq$
$\land\ Leftbank = \{\}$
$\land\ Rightbank = Farmyard$

Modification History
Last modified Fri Feb 05 12:22:31 GMT 2021 by alunm
Created Fri Feb 05 10:26:08 GMT 2021 by alunm

4 Model

Behaviour Specification is an *Initial predicate and next-state relation* of

Initial predicate *Initial*

Next-state relation *Next*

Checking The completion predicate is checked as an invariant

Invariants
$\neg Complete$

5 Results

A summary of the numbers of states found by the model checking is shown below.

5.1 Statistics

States found for model as a whole

States Found	20
Distinct States	10

Number of next states found for the actions is:

Action	**States found**	**Distinct states**
Init (line 14)	2	2
MoveEmpty (line 41)	8	2
Next (line 48)	14	5
Next (line 49)	9	2

5.2 Complete violated

The completion invariant is violated. The error trace shows the following moves:

State	*Boat*	*Leftbank*	*Rightbank*	**Action**
1	"left"	{"Wolf", "Sheep", "Cabbage"}	{}	*Initial predicate*
2	"right"	{"Wolf", "Cabbage"}	{"Sheep"}	Next line 48
3	"left"	{"Wolf", "Cabbage"}	{"Sheep"}	MoveEmpty line 42
4	"right"	{"Wolf"}	{"Sheep", "Cabbage"}	Next line 48
5	"left"	{"Wolf", "Sheep"}	{"Cabbage"}	Next line 49
6	"right"	{"Sheep"}	{"Wolf", "Cabbage"}	Next line 48
7	"left"	{"Sheep"}	{"Wolf", "Cabbage"}	MoveEmpty line 42
8	"right"	{}	{"Wolf", "Sheep", "Cabbage"}	Next line 48
				¬*Complete* violated